

Chapter 2

Multi-armed Bandits

The most important feature distinguishing reinforcement learning from other types of learning is that it uses training information that *evaluates* the actions taken rather than *instructs* by giving correct actions. This is what creates the need for active exploration, for an explicit search for good behavior. Purely evaluative feedback indicates how good the action taken was, but not whether it was the best or the worst action possible. Purely instructive feedback, on the other hand, indicates the correct action to take, independently of the action actually taken. This kind of feedback is the basis of supervised learning, which includes large parts of pattern classification, artificial neural networks, and system identification. In their pure forms, these two kinds of feedback are quite distinct: evaluative feedback depends entirely on the action taken, whereas instructive feedback is independent of the action taken.

In this chapter we study the evaluative aspect of reinforcement learning in a simplified setting, one that does not involve learning to act in more than one situation. This *nonassociative* setting is the one in which most prior work involving evaluative feedback has been done, and it avoids much of the complexity of the full reinforcement learning problem. Studying this case enables us to see most clearly how evaluative feedback differs from, and yet can be combined with, instructive feedback.

The particular nonassociative, evaluative feedback problem that we explore is a simple version of the k -armed bandit problem. We use this problem to introduce a number of basic learning methods which we extend in later chapters to apply to the full reinforcement learning problem. At the end of this chapter, we take a step closer to the full reinforcement learning problem by discussing what happens when the bandit problem becomes associative, that is, when actions are taken in more than one situation.

2.1 A k -armed Bandit Problem

Consider the following learning problem. You are faced repeatedly with a choice among k different options, or actions. After each choice you receive a numerical reward chosen from a stationary probability distribution that depends on the action you selected. Your

objective is to maximize the expected total reward over some time period, for example, over 1000 action selections, or *time steps*.

This is the original form of the *k*-armed bandit problem, so named by analogy to a slot machine, or “one-armed bandit,” except that it has k levers instead of one. Each action selection is like a play of one of the slot machine’s levers, and the rewards are the payoffs for hitting the jackpot. Through repeated action selections you are to maximize your winnings by concentrating your actions on the best levers. Another analogy is that of a doctor choosing between experimental treatments for a series of seriously ill patients. Each action is the selection of a treatment, and each reward is the survival or well-being of the patient. Today the term “bandit problem” is sometimes used for a generalization of the problem described above, but in this book we use it to refer just to this simple case.

In our k -armed bandit problem, each of the k actions has an expected or mean reward given that that action is selected; let us call this the *value* of that action. We denote the action selected on time step t as A_t , and the corresponding reward as R_t . The value then of an arbitrary action a , denoted $q_*(a)$, is the expected reward given that a is selected:

$$q_*(a) \doteq \mathbb{E}[R_t | A_t = a].$$

If you knew the value of each action, then it would be trivial to solve the k -armed bandit problem: you would always select the action with highest value. We assume that you do not know the action values with certainty, although you may have estimates. We denote the estimated value of action a at time step t as $Q_t(a)$. We would like $Q_t(a)$ to be close to $q_*(a)$.

If you maintain estimates of the action values, then at any time step there is at least one action whose estimated value is greatest. We call these the *greedy* actions. When you select one of these actions, we say that you are *exploiting* your current knowledge of the values of the actions. If instead you select one of the nongreedy actions, then we say you are *exploring*, because this enables you to improve your estimate of the nongreedy action’s value. Exploitation is the right thing to do to maximize the expected reward on the one step, but exploration may produce the greater total reward in the long run. For example, suppose a greedy action’s value is known with certainty, while several other actions are estimated to be nearly as good but with substantial uncertainty. The uncertainty is such that at least one of these other actions probably is actually better than the greedy action, but you don’t know which one. If you have many time steps ahead on which to make action selections, then it may be better to explore the nongreedy actions and discover which of them are better than the greedy action. Reward is lower in the short run, during exploration, but higher in the long run because after you have discovered the better actions, you can exploit *them* many times. Because it is not possible both to explore and to exploit with any single action selection, one often refers to the “conflict” between exploration and exploitation.

In any specific case, whether it is better to explore or exploit depends in a complex way on the precise values of the estimates, uncertainties, and the number of remaining steps. There are many sophisticated methods for balancing exploration and exploitation for particular mathematical formulations of the k -armed bandit and related problems.

However, most of these methods make strong assumptions about stationarity and prior knowledge that are either violated or impossible to verify in applications and in the full reinforcement learning problem that we consider in subsequent chapters. The guarantees of optimality or bounded loss for these methods are of little comfort when the assumptions of their theory do not apply.

In this book we do not worry about balancing exploration and exploitation in a sophisticated way; we worry only about balancing them at all. In this chapter we present several simple balancing methods for the k -armed bandit problem and show that they work much better than methods that always exploit. The need to balance exploration and exploitation is a distinctive challenge that arises in reinforcement learning; the simplicity of our version of the k -armed bandit problem enables us to show this in a particularly clear form.

2.2 Action-value Methods

We begin by looking more closely at methods for estimating the values of actions and for using the estimates to make action selection decisions, which we collectively call *action-value methods*. Recall that the true value of an action is the mean reward when that action is selected. One natural way to estimate this is by averaging the rewards actually received:

$$Q_t(a) \doteq \frac{\text{sum of rewards when } a \text{ taken prior to } t}{\text{number of times } a \text{ taken prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \cdot \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}}, \quad (2.1)$$

where $\mathbb{1}_{predicate}$ denotes the random variable that is 1 if *predicate* is true and 0 if it is not. If the denominator is zero, then we instead define $Q_t(a)$ as some default value, such as 0. As the denominator goes to infinity, by the law of large numbers, $Q_t(a)$ converges to $q_*(a)$. We call this the *sample-average* method for estimating action values because each estimate is an average of the sample of relevant rewards. Of course this is just one way to estimate action values, and not necessarily the best one. Nevertheless, for now let us stay with this simple estimation method and turn to the question of how the estimates might be used to select actions.

The simplest action selection rule is to select one of the actions with the highest estimated value, that is, one of the greedy actions as defined in the previous section. If there is more than one greedy action, then a selection is made among them in some arbitrary way, perhaps randomly. We write this *greedy* action selection method as

$$A_t \doteq \arg \max_a Q_t(a), \quad (2.2)$$

where $\arg \max_a$ denotes the action a for which the expression that follows is maximized (again, with ties broken arbitrarily). Greedy action selection always exploits current knowledge to maximize immediate reward; it spends no time at all sampling apparently inferior actions to see if they might really be better. A simple alternative is to behave greedily most of the time, but every once in a while, say with small probability ε , instead

select randomly from among all the actions with equal probability, independently of the action-value estimates. We call methods using this near-greedy action selection rule ε -greedy methods. An advantage of these methods is that, in the limit as the number of steps increases, every action will be sampled an infinite number of times, thus ensuring that all the $Q_t(a)$ converge to $q_*(a)$. This of course implies that the probability of selecting the optimal action converges to greater than $1 - \varepsilon$, that is, to near certainty. These are just asymptotic guarantees, however, and say little about the practical effectiveness of the methods.

Exercise 2.1 In ε -greedy action selection, for the case of two actions and $\varepsilon = 0.5$, what is the probability that the greedy action is selected? \square

2.3 The 10-armed Testbed

To roughly assess the relative effectiveness of the greedy and ε -greedy action-value methods, we compared them numerically on a suite of test problems. This was a set of 2000 randomly generated k -armed bandit problems with $k = 10$. For each bandit problem, such as the one shown in Figure 2.1, the action values, $q_*(a)$, $a = 1, \dots, 10$,

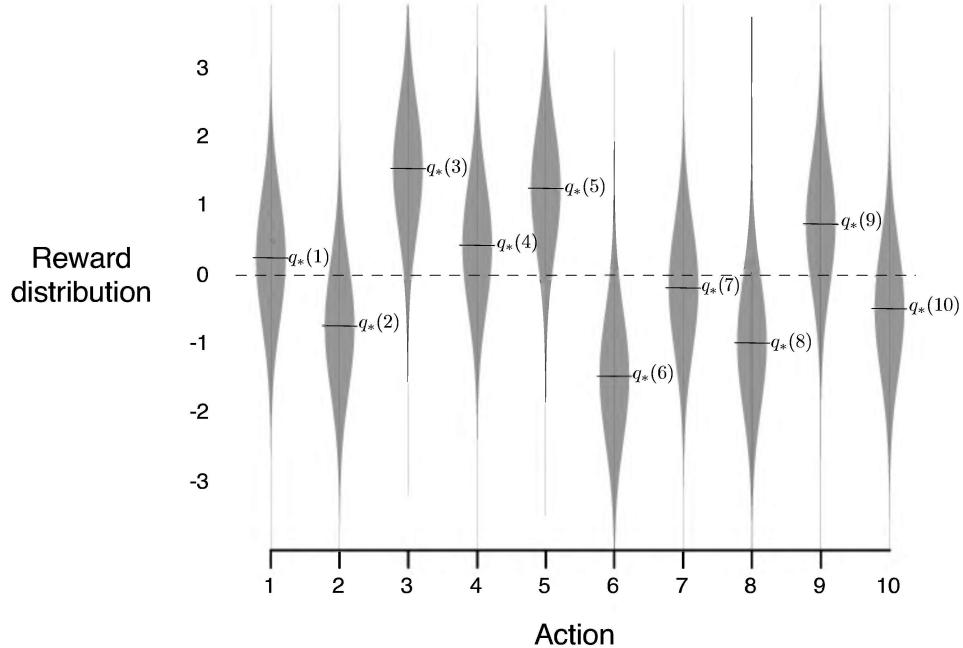


Figure 2.1: An example bandit problem from the 10-armed testbed. The true value $q_*(a)$ of each of the ten actions was selected according to a normal distribution with mean zero and unit variance, and then the actual rewards were selected according to a mean $q_*(a)$ unit variance normal distribution, as suggested by these gray distributions.

were selected according to a normal (Gaussian) distribution with mean 0 and variance 1. Then, when a learning method applied to that problem selected action A_t at time step t , the actual reward, R_t , was selected from a normal distribution with mean $q_*(A_t)$ and variance 1. These distributions are shown in gray in Figure 2.1. We call this suite of test tasks the *10-armed testbed*. For any learning method, we can measure its performance and behavior as it improves with experience over 1000 time steps when applied to one of the bandit problems. This makes up one *run*. Repeating this for 2000 independent runs, each with a different bandit problem, we obtained measures of the learning algorithm's average behavior.

Figure 2.2 compares a greedy method with two ε -greedy methods ($\varepsilon=0.01$ and $\varepsilon=0.1$), as described above, on the 10-armed testbed. All the methods formed their action-value estimates using the sample-average technique. The upper graph shows the increase in expected reward with experience. The greedy method improved slightly faster than the other methods at the very beginning, but then leveled off at a lower level. It achieved a reward-per-step of only about 1, compared with the best possible of about 1.55 on this testbed. The greedy method performed significantly worse in the long run because it

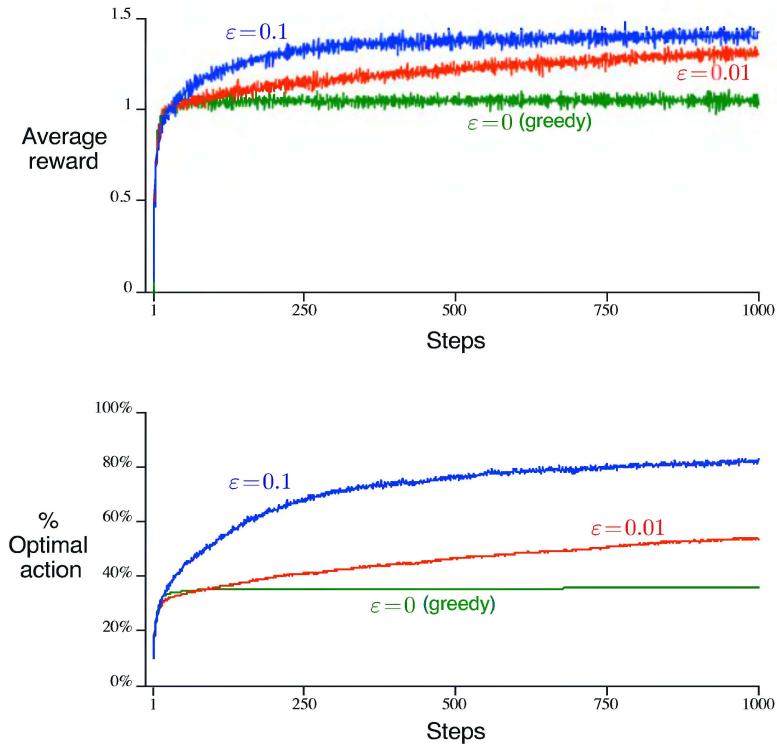


Figure 2.2: Average performance of ε -greedy action-value methods on the 10-armed testbed. These data are averages over 2000 runs with different bandit problems. All methods used sample averages as their action-value estimates.

often got stuck performing suboptimal actions. The lower graph shows that the greedy method found the optimal action in only approximately one-third of the tasks. In the other two-thirds, its initial samples of the optimal action were disappointing, and it never returned to it. The ε -greedy methods eventually performed better because they continued to explore and to improve their chances of recognizing the optimal action. The $\varepsilon = 0.1$ method explored more, and usually found the optimal action earlier, but it never selected that action more than 91% of the time. The $\varepsilon = 0.01$ method improved more slowly, but eventually would perform better than the $\varepsilon = 0.1$ method on both performance measures shown in the figure. It is also possible to reduce ε over time to try to get the best of both high and low values.

The advantage of ε -greedy over greedy methods depends on the task. For example, suppose the reward variance had been larger, say 10 instead of 1. With noisier rewards it takes more exploration to find the optimal action, and ε -greedy methods should fare even better relative to the greedy method. On the other hand, if the reward variances were zero, then the greedy method would know the true value of each action after trying it once. In this case the greedy method might actually perform best because it would soon find the optimal action and then never explore. But even in the deterministic case there is a large advantage to exploring if we weaken some of the other assumptions. For example, suppose the bandit task were nonstationary, that is, the true values of the actions changed over time. In this case exploration is needed even in the deterministic case to make sure one of the nongreedy actions has not changed to become better than the greedy one. As we shall see in the next few chapters, nonstationarity is the case most commonly encountered in reinforcement learning. Even if the underlying task is stationary and deterministic, the learner faces a set of banditlike decision tasks each of which changes over time as learning proceeds and the agent's decision-making policy changes. Reinforcement learning requires a balance between exploration and exploitation.

Exercise 2.2: Bandit example Consider a k -armed bandit problem with $k = 4$ actions, denoted 1, 2, 3, and 4. Consider applying to this problem a bandit algorithm using ε -greedy action selection, sample-average action-value estimates, and initial estimates of $Q_1(a) = 0$, for all a . Suppose the initial sequence of actions and rewards is $A_1 = 1$, $R_1 = -1$, $A_2 = 2$, $R_2 = 1$, $A_3 = 2$, $R_3 = -2$, $A_4 = 2$, $R_4 = 2$, $A_5 = 3$, $R_5 = 0$. On some of these time steps the ε case may have occurred, causing an action to be selected at random. On which time steps did this definitely occur? On which time steps could this possibly have occurred? \square

Exercise 2.3 In the comparison shown in Figure 2.2, which method will perform best in the long run in terms of cumulative reward and probability of selecting the best action? How much better will it be? Express your answer quantitatively. \square

2.4 Incremental Implementation

The action-value methods we have discussed so far all estimate action values as sample averages of observed rewards. We now turn to the question of how these averages can be computed in a computationally efficient manner, in particular, with constant memory

and constant per-time-step computation.

To simplify notation we concentrate on a single action. Let R_i now denote the reward received after the i th selection of this action, and let Q_n denote the estimate of its action value after it has been selected $n - 1$ times, which we can now write simply as

$$Q_n \doteq \frac{R_1 + R_2 + \cdots + R_{n-1}}{n-1}.$$

The obvious implementation would be to maintain a record of all the rewards and then perform this computation whenever the estimated value was needed. However, if this is done, then the memory and computational requirements would grow over time as more rewards are seen. Each additional reward would require additional memory to store it and additional computation to compute the sum in the numerator.

As you might suspect, this is not really necessary. It is easy to devise incremental formulas for updating averages with small, constant computation required to process each new reward. Given Q_n and the n th reward, R_n , the new average of all n rewards can be computed by

$$\begin{aligned} Q_{n+1} &= \frac{1}{n} \sum_{i=1}^n R_i \\ &= \frac{1}{n} \left(R_n + \sum_{i=1}^{n-1} R_i \right) \\ &= \frac{1}{n} \left(R_n + (n-1) \frac{1}{n-1} \sum_{i=1}^{n-1} R_i \right) \\ &= \frac{1}{n} \left(R_n + (n-1) Q_n \right) \\ &= \frac{1}{n} \left(R_n + n Q_n - Q_n \right) \\ &= Q_n + \frac{1}{n} [R_n - Q_n], \end{aligned} \tag{2.3}$$

which holds even for $n = 1$, obtaining $Q_2 = R_1$ for arbitrary Q_1 . This implementation requires memory only for Q_n and n , and only the small computation (2.3) for each new reward.

This update rule (2.3) is of a form that occurs frequently throughout this book. The general form is

$$\text{NewEstimate} \leftarrow \text{OldEstimate} + \text{StepSize} [\text{Target} - \text{OldEstimate}]. \tag{2.4}$$

The expression $[\text{Target} - \text{OldEstimate}]$ is an *error* in the estimate. It is reduced by taking a step toward the “Target.” The target is presumed to indicate a desirable direction in which to move, though it may be noisy. In the case above, for example, the target is the n th reward.

Note that the step-size parameter (*StepSize*) used in the incremental method (2.3) changes from time step to time step. In processing the n th reward for action a , the